REMARKS

Claims 1-16 and 20-33 were pending in the application at the time of the Office Action. Claims 1-16 and 20-33 were rejected under 35 U.S.C. 103. By this response, Applicant has amended claims 1, 2, 5-16, 20-22 and 25-29. Applicant respectfully submits that the amendment to the claims are based in the specification as originally filed and that no new matter has been added. Entry of the claim amendments is respectfully requested. As such, claims 1-16 and 20-33 are presented for the Examiner's consideration in light of the following remarks.

Reconsideration and allowance of the application is respectfully requested in view of the above amendments to the claims and the following remarks. Applicant requests that the Examiner carefully review any references discussed below to ensure that Applicant's understanding and discussion of the references, if any, is consistent with the Examiner's understanding. For the Examiner's convenience and reference, Applicant's remarks are presented in the order in which the corresponding issues were raised in the Office Action.

A. Examiner Telephone Interview

Applicant(s) and applicant's attorney express appreciation to the Examiner for the courtesies extended during the recent interview held on August 27, 2008. This response includes the substance of the Interview.

B. Rejection on the Merits

Independent claims 1, 20 and 27

Claims 1-5, 7, 10, 12-16, 20-25, 27, 29-32 were rejected under 35 U.S.C. 103(a) as being unpatentable over Beisiegel et al. (U.S. Pub No. 2004/0177335 A1) in view of Skrzynski et al. (U.S. Patent 6,691,302).

The present invention is related to a service-oriented software development system for developing and implementing service modules. The service-oriented software development system uses system functions to provide its functionality. System functions can include logging, management, memory sharing, caching, etc. At least some of the system functions of the service-oriented software development system are implemented as service-oriented software service modules. In addition, these system functions that are built as service-oriented software service modules can be run on the same platform that is provided by the service-oriented

<u>software development system</u>. In order to invoke a system function service-oriented software service module, the platform of the service-oriented software development system calls the system function service-oriented software service module and then implements it through <u>its own</u> platform. <u>Thus, the service-oriented software development system leverages its own service-oriented artifacts to perform its own system functionality.</u> *See* Specification, [para 3].

Notably, one aspect of the claims is that the service-oriented development system can be used to define its own system function service-oriented software service modules. For example, a developer needing a particular system function for the service-oriented development system uses the development system itself to define a new system function as a service-oriented software service module, and then implements the new service-oriented software service module using the same core module of the service-oriented development system. Because having a service-oriented development that is itself built on top of service-oriented service modules introduces circularity, the specification teaches that

> [Para 40] Any system or definition that references itself is circular by nature. For the purpose of the present invention, and to address this circularity, we define a portion of the system, hereon referred to as the Kernel, which breaks such circularity in avoiding infinitely recursive definitional references. Furthermore, we define the Kernel module as the core functional coordinator, software service dispatcher, and composition manager of the system of the present invention.

Thus, not only is the service-oriented development system able to be <u>used to define its own system functions</u> as service-oriented software service modules, it also becomes a <u>consumer</u> of these service-oriented software service modules. The specification further teaches:

> [Para 12] . . . the present invention provides methods, frameworks, and systems for building a software system for the management, implementation and composition of service-oriented, software modules <u>that are built themselves on top of service-oriented software modules leveraging its own artifacts</u> and set of functionality in management, implementation and composition of software modules. In preferred embodiments, this technique comprises: 1) dividing the system into a core module, the Kernel, that provides a basic framework utilizing software interfaces with plug-able implementations for the dispatch of software service module implementation with a multi-threaded process abstraction; <u>2) use of said framework by the Kernel itself and the management and design tools to provide the rest of the functionality that the system needs in performing its own tasks</u> including, but not limited to, metadata management, logging, cache management, system configuration, shared memory management, event broadcasting and notification,

> security and provisioning as a set of service modules dispatched by the Kernel.
>
> [Para 13] . . . The Graphical User Interface (GUI) portion of the system used for defining service interfaces, definition, composition and management tools, consumes "System" services to perform its function, while the implementation of the those (sic) services are dispatched by the Kernel and implemented through the same framework that is provided to the end-user for managing, implementing, deploying and developing any software service module. Each function for the system is first described using a set of service interface definitions using the system itself, while the GUI layer providing this function accesses the function through the consumption of software services based on the defined interfaces.

Applicants traverse the Examiner's rejection for obviousness on the grounds that the references – either individually or in combination – fail to teach or suggest each and every element of the rejected claims. By contrast to the presently claimed invention, neither Beisigel nor Skrzynski teach or suggest that the software development system is used to define or consume service modules to perform **its own** system functions as is presently claimed.

Rather, although Beisigel teaches "a service-oriented development model for enterprise applications and an integrated development environment for architecting such applications," (See Para. [0011]), the Examiner states, and Applicant agrees, that Beisigel does not teach or suggest "the system uses software service modules to perform system functions to enable operation of the system itself, wherein execution of the system functions includes the software service modules of the system functions being implemented through the server infrastructure itself." See Office Action, page 3.

Skrzynski addresses the problem where an application uses a service component to access system services of an operating system, but where the service component may be written in a different programming language than the native operating system API. For example,

> Operating systems, such as the Windows NT Server operating system, make extensive use of dynamic link libraries. A dynamic link library (DLL) is a computer code module that contains functions to be linked with application code. A DLL may be loaded and linked to an application at run time, and may be unloaded when its functionality is no longer needed. A DLL usually consists of a set of autonomous functions that any application may use. In 32-bit Windows operating systems, such as Windows NT, Windows 95 and Windows CE, all of the functions in the Win32 API are contained in DLLs . . . .
>
> . . . . most service components operating in a 32-bit Windows environment are written in the C or C++ programming languages . . . programs written in the

> JAVA programming language must operate in an environment provided by a JAVA virtual machine (JVM). The JVM is an abstract native computing machine that runs within an operating system to interpret and execute JAVA applications.

Skrzynski, col. 1, ll. 32-42, 46-49. Skrzynski then teaches

> the invention features an <u>interface module</u> configured to load a service component written in a non-native programming language . . . . The interface module preferably is configured to create an application operating environment in which the service component application is operable . . . .
>
> . . . .
>
> The invention provides a generic interface between a native operating system API and service components written in different programming languages (e.g., C, C++, and JAVA) . . . . Furthermore, new services may be readily added to the system without changing the dependability (sic) system. For example, with the invention a system designer merely has to replace the appropriate DLL or JAVA class with a new service module that implements the desired functionality.

Skrzynski at col. 2, ll. 2-13, 40-57 (emphasis added).

Thus, the interface modules taught in Skrzynski enable service components that would otherwise be incompatible with the native operating system API to be compatible with the native operating system API on which the service component and application are executed. Notably, Skrzynski teaches adding the interface modules onto already-existing service components so that the already-existing service components can communicate with the native operating system API. In other words, Skrzynski does not teach modifying the service components themselves, but rather, adding the interface modules to existing service components to communicate with the native operating system API. There is nothing in Skrzynski that teach or suggests that these service components (e.g., DLLs) are modified in any way to become service-oriented software modules, as that term is understood by those of skill in the art.

Thus, *at most*, Skrzynski teaches that if the "integrated development environment" application taught in Beisigel had service components that were written in a different language than the native operating system API, then interface modules such as those taught in Skrzynski could be used to allow the development environment service components to be operable with the native operating system API when the service component is programmed in a different language than the API.

However, Skrzynski does not teach or suggest

> wherein service interface definitions for the service-oriented software service modules that perform system functions are <u>first described using the set of management and design tools and then consumed by the same set of management and design tools</u>

as recited in independent claim 1;

> at least some of the system functions of the user interface tool and run-time server are implemented as service-oriented software service modules <u>using the user interface tool to define definitions of the system functions that are service-oriented software service modules</u>, wherein the <u>core module is configured to implement the at least some system functions by invoking the service-oriented software service modules of the at least some system functions</u>

as recited in independent claim 20; or

> <u>using a user-interface tool of a service-oriented development system to define interface definitions for one or more service-oriented software service modules for performing a system function</u> to enable operation of the service-oriented development system;
>
> using an invoker interface to request the one or more service-oriented software service modules <u>for performing a system function to enable operation of the service-oriented development system</u>, wherein, consuming a particular service-oriented software service module comprises generating an invocation request, and sending the invocation request to a core module

as recited in independent claim 27.

In view of both Beisiegel and Skrzynski's failure to teach at least these elements recited in independent claims 1, 20 and/or 27, Applicants submit that the Examiner has failed to set forth a *prima facie* case for obviousness and respectfully request that the rejection be withdrawn.

Dependent claims 2-5, 7, 10, 12-16, 21-25, and 29-32 depend from independent claims 1, 20 and 27 and thus incorporate the limitations thereof. As such, Applicant respectfully submits that claims 2-5, 7, 10, 12-16, 21-25, and 29-32 are distinguishable over the prior art for at least the same reasons discussed above with respect to claims 1, 20 and/or 27 and request that the obviousness rejection with respect to these claims be withdrawn.

Dependent claims 6, 8 and 9

Claims 6, 8 and 9 were rejected under 35 U.S.C. 103(a) over Beisiegel in view of Skrzynski et al., and further view of Bowman-Amuah (U.S. Pub No. 2001/0052108 A1).

Dependent claims 6, 8 and 9 depend from independent claim 1 and thus incorporate the elements thereof. As such, Applicant respectfully submits that claims 6, 8 and 9 are distinguishable over the prior art for at least the same reasons discussed above with respect to claim 1. Furthermore, the development architecture framework taught by Bowman-Amuah does not cure the deficiencies of Beisiegel/Skrzynki. As such, Applicant requests that the obviousness rejection with respect to these claims be withdrawn.

Dependent claims 11, 26, 28 and 33

Claims 11, 26, 28 and 33 were rejected under 35 U.S.C. 103(a) over Beisiegel et al. in view of Skrzynski et al. and further view of Lai (U.S. Pub No. 2005/0044197 A1).

Dependent claims 11, 26, 28 and 33 depend from independent claims 1, 20 and/or 27 and thus incorporate the elements thereof. As such, Applicant respectfully submits that claims 11, 26, 28 and 33 are distinguishable over the prior art for at least the same reasons discussed above with respect to claims 1, 20 and/or 27. Furthermore, the Web Services architecture taught by Lai does not cure the deficiencies of Beisiegel/Skrzynki. As such, Applicant requests that the obviousness rejection with respect to these claims be withdrawn.

C. Conclusion

In view of the foregoing, and consistent with the tentative agreement reached during the Examiner Interview, Applicants believe the claims as amended are in allowable form.

Applicant notes that this response does not discuss every reason why the presented claims are distinguished over the cited prior art. Most notably, applicant submits that many if not all of the dependent claims are independently distinguishable over the cited prior art. Applicant has merely submitted those arguments which it considers sufficient to clearly distinguish the claims over the cited prior art.

In the event that the Examiner finds remaining impediment to a prompt allowance of this application that may be clarified through a telephone interview, the Examiner is requested to contact the undersigned attorney.

Dated this 27th day of August, 2008.

Respectfully submitted,

/sara d. jones/ Registration # 47,691
SARA D. JONES
Registration No. 47,691
Attorney for Applicant
Customer No. 022913

SDJ: vlr